1. Introduction et Objectifs

BrokerX+ est une plateforme émergente de courtage en ligne qui souhaite moderniser son système de gestion des opérations de courtage pour répondre à la croissance de sa clientèle et aux exigences de fiabilité, sécurité et performance.

Panorama des exigences

BrokerX est une plateforme de courtage en ligne pour investisseurs particuliers. Cette application offre une interface de courtage moderne. Cette première phase a pour but de :

- concevoir une architecture monolithique évolutive,
- appliquer les principes du Domain-Driven Design,
- mettre en oeuvre des patron de conceptions adaptés,
- concevoir une solution de persistance robuste et fiable,
- documenter et justifier les choix architecturaux,
- implémenter un prototype réaliste et fonctionnel,git push -u origin main
- assurer la qualité par les tests automatisés,
- mettre en place et enrichir les pratiques DevOps.

Objectifs qualité

Priorité	Objectif qualité	Scénario
1	**Testabilité**	Tests automatisés avec pytest pour toutes les fonctions
2	**Déployabilité**	Pipeline CI/CD automatisé avec GitLab/GitHub
3	**Maintenabilité**	Code simple et bien structuré pour faciliter l'évolution

Parties prenantes (Stakeholders)

- Clients : utilisateurs via interface web/mobile.
- Opérations Back-Office : gestion des règlements, supervision.
- Conformité / Risque : surveillance pré- et post-trade.
- Fournisseurs de données de marché : cotations en temps réel.
- Bourses externes : simulateurs de marché pour routage d’ordres.

2. Contraintes

Contrainte	Description
Technologie	Utilisation de Java/C++/Rust/... (pas de Python), Docker, et GitLab/GitHub CI/CD
Déploiement	Déploiement via conteneur Docker et pipeline GitLab/GitHub
PostgreSQL	Base de données relationnelle
Monolithe initial	Architecture monolithique Phase 1

3. Contexte et champ d'application

3.1 Contexte métier

Le système permet au client de :

- créer un compte
- approvisionner son portfeuille
- faire des transaction

3.2 Contexte technique

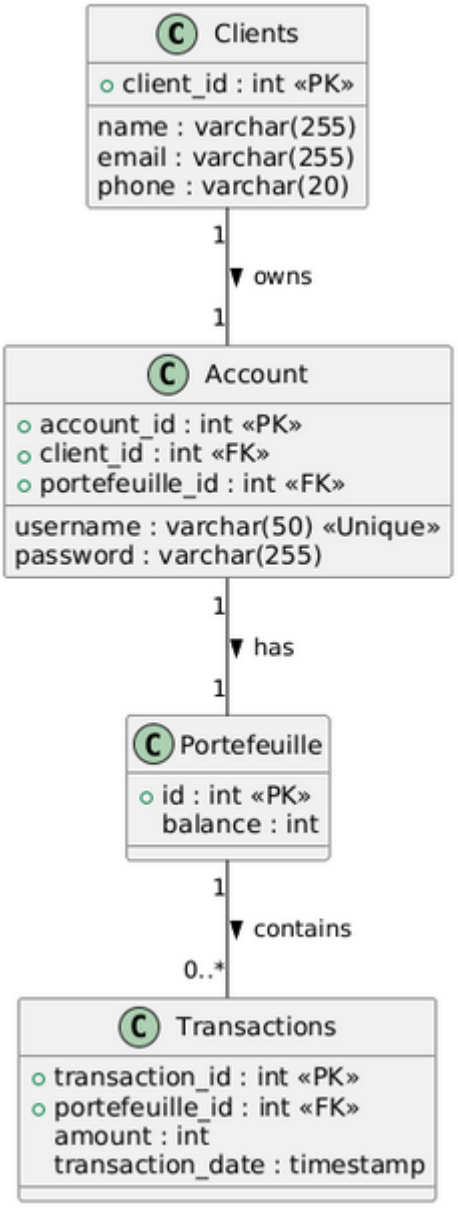
- **Client** : *main.rs* - Application Rust CLI
- **Couche base de données** : Backend PostgreSQL
- **Communication** : Communication direct entre l'application Rust et la base de données, via la librairie diesel (pas de couche API HTTP)

4. Stratégie de solution

5. Vue du bloc de consctruction

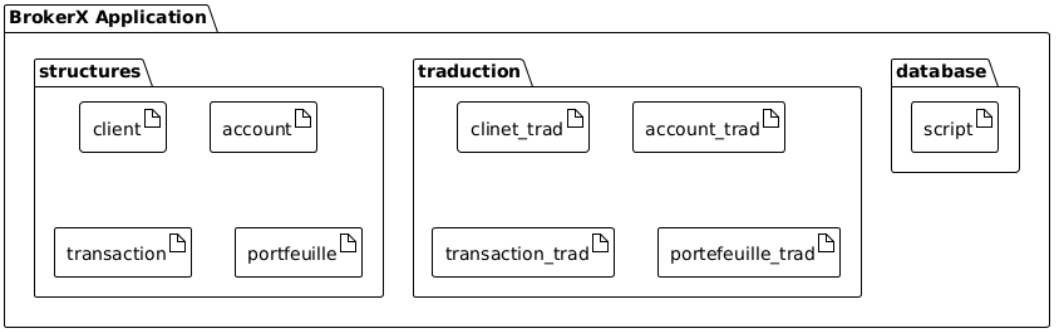
5.1 Diagramme de classes

Cette vue présente les principaux éléments métiers du système, ainsi que leurs relations. Dans cette première phase, on se concentre sur trois cas d'utilisation prioritaires : inscription, authentification et approvisionnement du portefeuille. Cela se traduit par deux grands ensembles : la gestion des clients et la gestion des portefeuilles.

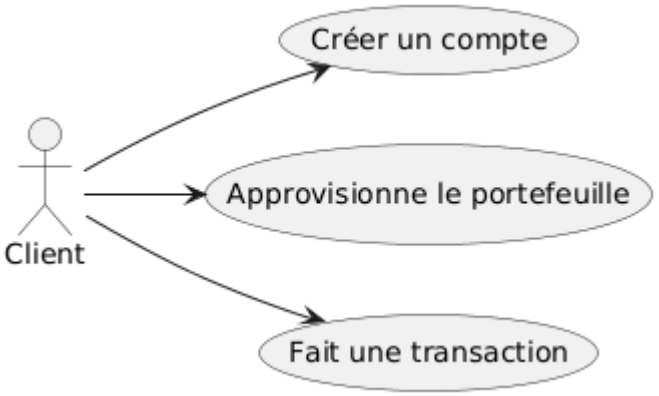


Ici, le client est l'élément central : il possède un compte pour s'authentifier et un portefeuille pour stocker ses fonds. Chaque portefeuille contient une suite de transactions qui permettent de retracer son historique.

5.2 Diagramme de paquetage

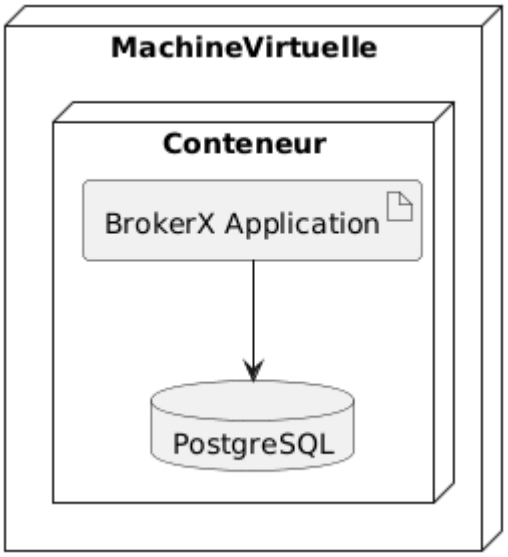


6. Vue d'exécution



7. Vue de déploiement

Cette vue montre comment le système est déployé techniquement. On utilise Docker pour exécuter l'application et sa base de données.



8. Concepts transversaux

- Patron client-serveur, ORM, Architecture hexagonal
- Persistance, base de donnée relationelle

9. Décisions architecturales

ADR001 - Style d'architecture

ADR002 - Choix de la persistance

ADR003 - Gestion des erreurs et idempotence des opérations

10. Exigence de qualité

Maintenabilité

- Séparation claire des responsabilités via Architecture hexagonal+ORM
- Conventions de nommage cohérentes à travers toutes les couches

Flexibilité

Evolutivité

- L'application peut avoir plusieurs clients connectés à un serveur
- L'application peut également avoir plusieurs serveurs, même s'ils ne partagent pas les mêmes données

11. Risque et dette techniques

12. Glossaires

Glossaire Métier

Terme	Définition
Back-Office	Équipe ou système gérant les opérations internes de règlement, supervision et conformité.
Client	Utilisateur de la plateforme
Compte	Identité numérique du client, utilisée pour l'inscription, l'authentification et la gestion de son portefeuille.
Conformité / Risque	Processus de surveillance des transactions avant (pré-trade) et après (post-trade) leur exécution.
Fournisseurs de données de marché	Entités fournissant des cotations en temps réel (prix des actifs).
Gestion de portefeuille	Fonctionnalité permettant au client d'approvisionner et de suivre ses fonds.
Portefeuille	Compte financier détenu par un client, contenant des fonds et des transactions.
Transaction	Opération financière réalisée par un client (ex. dépôt, retrait, achat/vente d'actif).

Stakeholder (Partie prenante)	Acteur ayant un intérêt ou une influence dans le projet

Glossaire Technique

Terme	Définition
ADR (Architectural Decision Record)	Document décrivant une décision architecturale importante et sa justification.
Architecture hexagonale	Modèle favorisant la séparation entre le domaine métier, les interfaces et les systèmes externes.
CI/CD	Pratiques d'intégration et déploiement continus, automatisant tests et livraison logicielle.
Conteneur Docker	Environnement isolé pour exécuter des applications et leurs dépendances.
Contrainte technologique	Limitation imposée au projet (Rust, Docker, PostgreSQL, etc.).
Diesel (ORM)	Librairie Rust pour interagir avec une base de données relationnelle.
Domain-Driven Design (DDD)	Méthodologie structurant le code autour du domaine métier et de ses règles.
Évolutivité	Capacité d'un système à croître sans perte de performance.
Flexibilité	Capacité d'un système à s'adapter rapidement aux évolutions.
GitLab/GitHub	Plateformes de gestion de code source et de pipelines CI/CD.
Idempotence	Propriété d'une opération répétée sans effet supplémentaire.
Maintenabilité	Facilité à modifier, améliorer ou corriger le code.
Monolithe	Architecture où toutes les fonctionnalités sont regroupées dans une seule application.
ORM (Object-Relational Mapping)	Technique de manipulation d'une base relationnelle via des objets du langage.
Patron de conception	Solution réutilisable à un problème de conception logiciel récurrent.

Pipeline	Suite d'étapes automatisées pour tester, intégrer et déployer du code.
PostgreSQL	Système de gestion de base de données relationnelle.
Sécurité	Mesures de protection des données et transactions des clients.
Testabilité	Capacité à écrire et exécuter facilement des tests automatisés.